

Operating System Support for Fine-Grained Task Migration

Roberto Gioiosa
Pacific Northwest National Lab
Richland, WA 99352
Email: roberto.gioiosa@pnnl.gov

Sriram Krishnamoorthy
Pacific Northwest National Lab
Richland, WA 99352
Email: sriram@pnnl.gov

INTRODUCTION

With the end of clock scaling and the limited power budget available (20-30MW), future supercomputers will meet exascale performance primarily through a higher level of parallelism. Current operating (OS) and runtime (RT) systems are designed for the classical SMP model and based on the static and coarse-grained process/thread paradigm. They do not provide the required level of flexibility, especially within a single compute node, to meet the requirements imposed by exascale systems in terms of power/energy efficiency, resilience, managing concurrency and performance portability.

The high level of concurrency poses new challenges specific to exascale systems that need to be addressed by novel solutions. In particular, requiring the user to manage billions of concurrent threads could easily result in poor data locality, clogged interconnection networks, unchecked propagation of soft errors, and lack of control over power/energy consumption. Equally important, managing such level of concurrency interferes with the user's focus on the application and the algorithm.

We envision that the computation will be encapsulated into fine-grained tasks that can be isolated and protected from the other tasks running in the system. Whenever a task needs to work on some data that is not stored on the local node, the OS/RT allows the task to be migrated to the node that owns the data. Each task is associated with a contained state (set or architectural registers, stack frame, running node, etc.) that describes the progress of the task and that should be moved together with the task's code. We believe system support for task migration is a fundamental function that can ease the job of tackling several of the exascale challenges.

This support contrasts with the current message-passing models, in which data is moved to the node where computation is preformed — an approach that strives to minimize reliance on the operating system [1]. In more detail, support for fine-grained task migration has a direct impact on the following:

a) Performance: OS/RT support for fine-grained task migration enables automated mechanisms to tolerate variations in the execution environment that result in load imbalance. Whether the load imbalance is induced by faults (i.e., some tasks have to be re-executed because the experienced an error in the previous execution), localized power considerations, or changes in the application's parallelism profile, tasks can be migrated from heavily loaded nodes to nodes where there is availability of computing resources. This low-level feature well suites the high-level, task-based programming models, such as Cilk [2], [3], Chapel [4], and Charm++ [5]. In particular, while stack-less tasks have traditionally been supported through compiler transformations [3], [6], efficient support for tasks with stack state requires operating system support. Rather than rely on specific programming models for the exascale, an unforeseeable challenge, it is more reasonable to provide the fundamental constituent functionality that will be essential to build them. We believe OS support for task-migration can perform the basis for a broad set of exascale programming models that support fine-grained concurrency.

b) Power/Energy: The data footprint of exascale applications is expected to be considerably larger than today's applications. In this scenario, current OS/RT models based on message-passing require data exchange among the application's tasks, which has a direct impact on the total system power consumption. Conversely, since the application's code is duplicated on all compute nodes, only the task's state and the input parameters need to be transferred to a remote node. We can quantify these movements in the order of tens/hundreds of kilobytes as opposed to movements of hundreds of megabytes/gigabytes required to transfer data, with proportional power and energy savings.

c) Resilience: Fine-grained task migration is inherently fault tolerant. Migrating a task from one node to another requires saving the task's state on the origin node and moving it to a destination node. This is equivalent to taking a checkpoint of the task and eventually restarting it on a remote node in case of a fault.

Should a task experience a fault during its execution, the OS/RT simply restarts the task from the last valid state (the one transferred to the local node or saved on the origin node) potentially with little intervention from the programmer. Once the execution of a task is marked as completed and reliable, the new state can be saved (checkpointed) and used as the last valid state in the ensuing computation. This approach enables fine-grained, reliable checkpointing without expensive global synchronization.

d) Locality: Exploiting locality is another of the characteristics intrinsic to the fine-grained task migration model. Since computation can be moved to where the data is stored and since the tasks' state is orders of magnitude smaller than the application's data footprint, data movement is reduced to the task's parameters required to perform computation on the destination node.

CHALLENGES ADDRESSED

We address several key exascale challenges. The ability to save and migrate task state enables efficient load-balanced execution of a broad class of programming models, such as Charm++ [5], qthreads [7], Chapel tasks [2], [3], that represent computation as consisting of a disjoint unit of control, data (possibly on heap), and stack state. Task migration also enables the design of programming languages and runtimes to tolerate variations due to power/energy constraints or faults. The availability of this functionality at the fundamental OS level also enables performance portability of a broad class of applications on emerging architectures, without requiring repeated rewriting of applications and runtimes.

MATURITY AND NOVELTY

There has been work on programming model and architecture support for task migration in a variety of contexts [3]–[6], [8]–[11]. However, these efforts are limited by the fact that current OSs do not support fine-grained task migration efficiently, which we believe it is fundamental for exascale systems. These research experiences lead us believe that efficient OS/RT support for fine-grained task migration will be feasible and broadly adopted.

UNIQUENESS

Existing commercial approaches that employ task migration, such as Hadoop and MapReduce, do not try to address this problem because they are mainly design for different workload/scenarios where performance and efficiency are not of paramount importance.

APPLICABILITY

A successful implementation of efficient OS/RT support for task migration will strongly influence the design of programming models that explore fine-grained concurrency. This effort will also influence and guide architectural support for micro-checkpointing, transactional memory, thread-level speculation, etc.

EFFORT

Given the influence of this effort on architectural design and programming models, we believe the effectiveness of this approach, in the form of prototypes, needs to be proved early in the exascale roadmap. This could be achieved through multiple small coordinated efforts that explore competing approaches to fine-grained task migration in the OS/RT.

REFERENCES

- [1] P. Shivam, P. Wyckoff, and D. Panda, "Emp: zero-copy os-bypass nic-driven gigabit ethernet message passing," in *Supercomputing, ACM/IEEE 2001 Conference*. IEEE, 2001, pp. 49–49.
- [2] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 55–69, August 25 1996. [Online]. Available: <ftp://theory.lcs.mit.edu/pub/cilk/cilkjpd96.ps.gz>
- [3] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the Cilk-5 multithreaded language," in *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI)*, Montreal, Quebec, Canada, Jun. 1998, pp. 212–223, proceedings published ACM SIGPLAN Notices, Vol. 33, No. 5, May, 1998.
- [4] B. Chamberlain, D. Callahan, and H. Zima, "Parallel programmability and the chapel language," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 291–312, Aug. 2007. [Online]. Available: <http://dx.doi.org/10.1177/1094342007078442>
- [5] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of OOPSLA'93*, A. Paepcke, Ed. ACM Press, September 1993, pp. 91–108.
- [6] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. Von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," in *ACM SIGPLAN Notices*, vol. 40, no. 10. ACM, 2005, pp. 519–538.
- [7] K. B. Wheeler, R. C. Murphy, and D. Thain, "Qthreads: An api for programming with millions of lightweight threads," in *IPDPS*, 2008, pp. 1–8.
- [8] R. Agarwal, P. Garg, and J. Torrellas, "Rebound: scalable checkpointing for coherent shared memory," in *ISCA*, 2011.
- [9] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, "Ibm power7 multicore server processor," *IBM J. Res. Dev.*, vol. 55, no. 3, pp. 191–219, May 2011. [Online]. Available: <http://dx.doi.org/10.1147/JRD.2011.2127330>
- [10] A. Chakravarti, X. Wang, J. Hallstrom, and G. Baumgartner, "Implementation of strong mobility for multi-threaded agents in java," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*. IEEE, 2003, pp. 321–330.
- [11] L. Bettini and R. De Nicola, "Translating strong mobility into weak mobility," in *Mobile Agents*, ser. Lecture Notes in Computer Science, G. Picco, Ed. Springer Berlin / Heidelberg, 2001, vol. 2240, pp. 182–197.